TemporalMaxer Performance in the Face of Constraint: A Study in Temporal Action Localization

A Comprehensive Analysis on the Adaptability of TemporalMaxer in Resource-Scarce Environments

Teodor-Gabriel Oprescu[1]

Supervisors: Dr. Jan van Gemert[1]**, Robert-Jan Bruintjes**[1]**,**
Attila Lengyel[1]**, Ombretta Strafforello**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Teodor-Gabriel Oprescu
Final project course: CSE3000 Research Project
Thesis committee: Dr. Jan van Gemert, Ombretta Strafforello, Robert-Jan Bruintjes, Attila Lengyel, Dr.-Ing. Petr Kellnhofer

Abstract

This paper presents an analysis of the data and compute efficiency of the TemporalMaxer deep learning model in the context of temporal action localization (TAL), which involves accurately detecting the start and end times of specific video actions. The study explores the performance and scalability of the TemporalMaxer model under limited resources and data availability, focusing on factors such as hardware requirements, training time, and data utilization, thus contributing to the advancement of efficient deep learning models for real-world video tasks. Through a literature review of temporal action recognition models, evaluation of learning curves for data efficiency, and development of metrics to assess the compute efficiency, the study provides insights into the performance trade-offs of the TemporalMaxer model. Experiments conducted on the widely used THUMOS dataset further demonstrate the model's generalizability with limited data, achieving significant accuracy performance with only 50% of the training data. Notably, TemporalMaxer exhibits superior compute efficiency by significantly reducing the number of Multiply-Accumulate operations (MACs) compared to other state-of-the-art models. However, alternative models like TriDet and TadTR outperform TemporalMaxer in training time-constrained scenarios. These findings shed light on the model's practical applicability in resource-constrained environments, offering insights for further optimization and study.

1. Introduction

Temporal action localization (TAL) is a crucial task in video analysis, aiming to detect and localize specific actions within a video sequence. It has extensive applications in various domains, including video surveillance [1], video summarization [2], skill assessment [3], and action retrieval [4, 5], thus receiving widespread attention from both the academia and the industry in recent years. With the rise of deep learning techniques, TAL models have witnessed significant advancements, which excel at learning hierarchical representations from video data.

Despite considerable advancements, TAL still presents substantial challenges posed by the complex spatiotemporal backgrounds, ambiguous temporal boundaries, and significant variations in people's appearances, camera viewpoint, and action duration [6]. Moreover, one key challenge is the demand for large-scale labelled video datasets needed to train these models effectively. Collecting and annotating these datasets is time-consuming, costly, and often limited by resource availability and expertise [7]. However, training deep neural networks also requires substantial computational resources and time. This situation restricts the accessibility of state-of-the-art (SOTA) TAL models to a limited number of well-funded entities. While the straightforward solution might seem to be simply adding more data to the model's training to counter these issues, this approach quickly becomes unfeasible since it only increases the already high computational costs and time requirements. To address this issue, the THUMOS [8] initiative has emerged as a vital contributor, gaining widespread adoption within the TAL community and becoming a benchmark dataset.

In the Temporal Action Localization (TAL) field, several approaches have been proposed to achieve high performance in identifying and localizing actions within video sequences. Some of the most-performing approaches include those that use complex self-attention mechanisms [9], transformer architectures [10], and long-term temporal context modelling [11]. One of the state-of-the-art (SOTA) models in TAL is ActionFormer [10], which follows a minimalistic design of sequence labelling, where each moment is classified, and the corresponding action boundaries are regressed. It employs a transformer-based architecture to capture long-term dependencies and temporal context in the video clips [10].

Taking inspiration from ActionFormer [10], the TemporalMaxer method is introduced [12] that presents a novel approach that aims to maximize temporal context while minimizing the complexity of the model [12]. It uses a basic, parameter-free, and local region operating max-pooling block. This block selectively picks out the most critical information from adjacent and local clip embeddings, resulting in a more efficient TAL model. TemporalMaxer outperforms other SOTA methods that rely on long-term temporal context modelling, such as self-attention models, while requiring significantly fewer parameters and computational resources [13].

This research addresses the challenges of data and compute efficiency in TAL by closely investigating the performance and scalability of the TemporalMaxer [12] deep learning model under limited compute resources and data availability. The goal is achieving accurate action localization while mitigating the demands for extensive datasets and compute power. The THUMOS14 dataset [8] will serve as a benchmark for assessing the impact of these constraints on the model's accuracy. To achieve this objective, we will compare the results with those reported in the original TemporalMaxer introductory paper [12] by making use of the DelftBlue supercomputer [14]. Moreover, we will analyze the data and compute efficiency of TemporalMaxer against other influential TAL models such as ActionFormer [10,15], TriDet [16, 17]. and TadTR [18, 19]. Through this analysis, we aim to provide insights into the trade-offs between model complexity and performance under resource constraints. These insights have the potential to speed up the research process in the TAL field, allowing researchers to assess the efficiency of an approach without the need for comprehensive training and testing.

2. Related work

2.1. Temporal Action Localization

Temporal action localization represents the problem of detecting the start and end times of actions in untrimmed videos, where the videos contain multiple actions with varying lengths and occur at different times. The untrimmed video, denoted as X, can be represented by a collection of feature vectors $X = \{x_1, x_2, ..., x_\mathrm{T}\}$, where the number of discrete time steps, denoted as $t = \{1, 2, ...T\}$, can vary depending on the video's length. Each feature vector x_t is obtained by extracting information from a pre-trained 3D convolutional network, representing a specific moment in the video at time t [12].

The goal of TAL is to predict a set of action instances $\psi = \{\psi_1, \psi_2, ...\psi_\mathrm{N}\}$ based on the input video sequence X, with N representing the number of action instances present in X. Each action instance ψ_n consists of three components: the starting time (s_n), the ending time (e_n), and the associated action label (a_n), with $s_n \in [1, T]$, $e_\mathrm{n} \in [1, T]$ and $s_\mathrm{n} < e_\mathrm{n}$. The action label a_n belongs to a pre-defined set of C categories [12].

Temporal Action Localization (TAL) employs both Two-Stage and Single-Stage methods to detect actions in videos [20]. Two-Stage methods [21] generate action proposals and then classify them through anchor windows, action boundaries detection, graph representation, or Transformers. Single-Stage TAL [22, 23] performs action proposal generation and classification in one pass, with notable examples including anchor-based and anchor-free single-stage models.

TAL models often strive to capture long-term temporal dependencies within video clip features, usually utilizing features from 3D-CNN networks as input [24, 25]. These models generally comprise an encoder or backbone and a decoder or head. To enhance the capture of long-term dependencies, long-term temporal context modelling (TCM) blocks are integrated into the backbone, with different techniques including Graph [26], Local-Global Temporal Encoder [12], and Relation-aware Pyramid Network [27] being explored. Despite the notable improvements achieved through self-attention for long-term TCM, there are concerns about inference speed and effectiveness. TemporalMaxer capitalizes on the strong features of the pre-trained 3D CNN and focuses on efficiently utilizing a Max Pooling block for local context modelling [12].

2.2. Data Efficiency

In Temporal Action Localization, data efficiency pertains to a model's capability to learn from limited labelled training data, which is essential due to the resource-intensive nature of obtaining large-scale datasets. A range of approaches can be deployed to augment data efficiency.

Transfer learning [28] pretrains models on substantial datasets and fine-tunes them on smaller TAL datasets, leveraging learned features to lessen extensive training on the target dataset. Semi-supervised learning [29] enhances generalizable representations by combining labelled and unlabeled data, while active learning employs a selective annotation of the most informative unlabeled samples. This iterative process enhances performance while limiting annotation efforts. Weakly-supervised learning [30] reduces the need for detailed frame-level annotations by employing weak annotations for training. Challenges such as limited dataset availability, domain shift, balancing model complexity and interpretability, and trade-offs between annotation effort and performance persist despite these methodologies.

Techniques like few-shot learning, zero-shot learning, and learning curve analysis are used to evaluate data efficiency. Few-shot learning [31], which involves training models on a minimal number of samples per class, tests the model's ability to generalize. Zero-shot learning [32] assesses the model's capabilities to recognize unseen classes, reflecting its generalization capabilities. Learning curves, which plot accuracy against the volume of training data, provide insights into performance shifts with increasing data availability.

The study uses the approach of sampling percentages of training data due to class imbalances and the presence of multiple classes in single videos. The practicality of a few-shot learning approach was seen as challenging within the project's timeframe.

2.3. Compute Efficiency

In Temporal Action Localization, compute efficiency relates to a model's ability to accurately localize action while minimizing computational resources, such as Multiply-Accumulate operations (MACs) and inference time. This involves models that efficiently process videos and perform action localization tasks with fewer computational demands.

Several methods to improve TAL models' compute efficiency include creating lightweight model architectures tailored for TAL tasks, applying temporal downsampling techniques like adaptive frame skipping [33], using temporal pooling operations to reduce temporal resolution [34], implementing pruning and quantization techniques [35], and employing knowledge distillation [36] to train a more computationally efficient model.

However, challenges arise in achieving compute efficiency, including the trade-off between model complexity and performance, balancing accuracy and compute efficiency and hardware limitations. These are particularly relevant in real-time action localization applications, where timing constraints are strict.

Measures such as the number of floating point & MAC operations, memory used, and inference and training times are used to test for computational efficiency. For instance, TAL models such as TriDet [16], TemporalMaxer [12], and ActionFormer [10] report MAC operations and the time to forward a single video. Nonetheless, no experiments have assessed how these models scale with increasing video length. Therefore, this study measures TemporalMaxer's inference performance on videos of varying lengths, recording the inference time, VRAM used, and the number of MAC operations executed.

Efforts to optimize for computational efficiency in TAL models, such as the low computational cost designs of TriDet [16] and TemporalMaxer [12], aims to enable effective action localization on resource-constrained devices or situations with limited computational resources. Through these optimization techniques and striking the right balance between efficiency and accuracy, TAL models can offer effective action localization with reduced computational demands.

3. Methodology

3.1. TemporalMaxer

TemporalMaxer [12] is a proposed method for temporal action localization (TAL), which aims to accurately detect and localize action instances in videos (visualization of the model is presented in Figure 1). The model focuses on extreme minimization of the backbone network by maximizing the information from extracted video clip features in a short-term perspective. Instead of employing complex modules or attention mechanisms, TemporalMaxer uses a basic, parameter-free, local operating Max Pooling block as its temporal context modelling (TCM) component. The model takes pre-extracted features from a 3D convolutional neural network (CNN) as input and encodes them using the simplified backbone. By leveraging the strong features of the pre-trained 3D CNN, TemporalMaxer maintains only the most critical information on adjacent and local clip embeddings. Combining the simplified backbone and the large receptive field of deep networks enables TemporalMaxer to outperform other TAL methods by a sensible margin regarding accuracy and speed on various challenging datasets, including THUMOS'14 [37].

3.2. Data Efficiency

Chosen metrics. In the context of evaluating TAL deep learning models like TemporalMaxer, the most common metric for evaluation is the mean average precision (mAP), which is evaluated across different Intersection over Union (tIoU) thresholds [7]. Mean average precision (mAP) combines precision and recall in a single metric. Precision (the ratio of correctly predicted positive instances to the total predicted positives) gauges the model's accuracy. However, precision doesn't consider the missed positives or false negatives, so recall is introduced.

mAP calculates the mean precision at different recall levels, ranging from 0 to 1, for each class of action c of a video. It begins by sorting all the predicted class instances by their confidence scores, then calculates precision and recall every time a positive instance is encountered. The precision values are then interpolated across the full range of recall levels to create a precision-recall curve. The averages of these interpolated precision values give the average precision (AP) for that class, and by computing the mean of the AP values across all classes, the mAP is obtained. It effectively integrates model performance across different classes and varying confidence levels, presenting a balanced perspective of its effectiveness. A high mAP score signifies a model that is accurate in its predictions (high precision) and covers a large portion of actual positive instances (high recall).

The tIoU (Intersection over Union) measures the overlap between the predicted and actual duration of an action in a video. It is calculated as the ratio of the intersection (the duration where both the predicted and actual action exists) to their union (the combined duration of the predicted and actual action). For any tIoU threshold, μ, and a class, c, correct predictions are those with a tIoU equal to or exceeding μ, and where the predicted class matches the actual class, c.

Evaluation. We will use Algorithm 1, previously introduced in [15] for evaluation of the data efficiency of the model. We divide the dataset $\mathcal{D}$ into a training set ($\mathcal{D}_{\text{train}}$) and a testing set ($\mathcal{D}_{\text{test}}$), according to THUMOS'14 specifications [8]. A percentage p of $\mathcal{D}_{\text{train}}$ is randomly selected to create a new set, $\mathcal{D}_{\text{s}}$. This set $\mathcal{D}_{\text{s}}$ is then used to train the model, after which its performance is assessed on $\mathcal{D}_{\text{test}}$. The mAP is then measured at different tIoU thresholds during this evaluation. This full sampling, training, and testing process is performed five times, each time with varying random splits. Subsequently, the average for each mAP threshold is determined, and the standard deviation is recorded. The whole procedure is repeated for different percentages of p.

The sampling method, as depicted in the pseudocode of Algorithm 1, randomly selects videos from the training set such that the size of $\mathcal{D}_{\text{s}}$ is rounded to the nearest integer when ($\mathcal{D}_{\text{train}}$) is multiplied by p over 100%. Furthermore, the sampling method must ensure that every action class is represented at least once in the resulting set $\mathcal{D}_{\text{s}}$. In a practical scenario, this was achieved by continually sampling from $\mathcal{D}_{\text{train}}$ until a split was found that represented all classes. The calculate-mAP method assesses the model's performance by calculating the mean average precision (mAP) at various tIoU thresholds that the model attained on the test set $\mathcal{D}_{\text{test}}$.

Algorithm 1 Data efficiency evaluation procedure

```
$\mathcal{D} = \{(\mathbf{V_i}, \mathbf{y_i})\}_{i=1}^{N}$
$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}} \leftarrow \texttt{split}(\mathcal{D})$
for $p$ in $[10\%, 20\%, 40\%, 60\%, 80\%, 100\%]$ do
    mAPs $\leftarrow$ empty list
    for $i = 1, ..., 5$ do
        $\mathcal{D}_{\text{s}} \leftarrow \texttt{sample}(\mathcal{D}_{\text{train}}, p)$
        Train on $\mathcal{D}_{\text{s}}$
        mAP $\leftarrow \texttt{calculate-mAP}(\mathcal{D}_{\text{test}})$
        Append mAP to mAPs
    Report $\mu_{\text{mAPs}}$ and $\sigma_{\text{mAPs}}$
```

For extrapolation purposes to other datasets, we use the equation: instancesPerClass $= \frac{pM}{C}$; to report the average action instances per class for each percentage p of the dataset. Here, M represents the total action instances, and C is the number of distinct classes. This estimated count is used as the precise number could be skewed by the specific splits (D_{s}) used in the data efficiency experiment.

3.3. Compute Efficiency

3.3.1 Training efficiency

The training efficiency of the TemporalMaxer model was analyzed by recording its training duration and the average

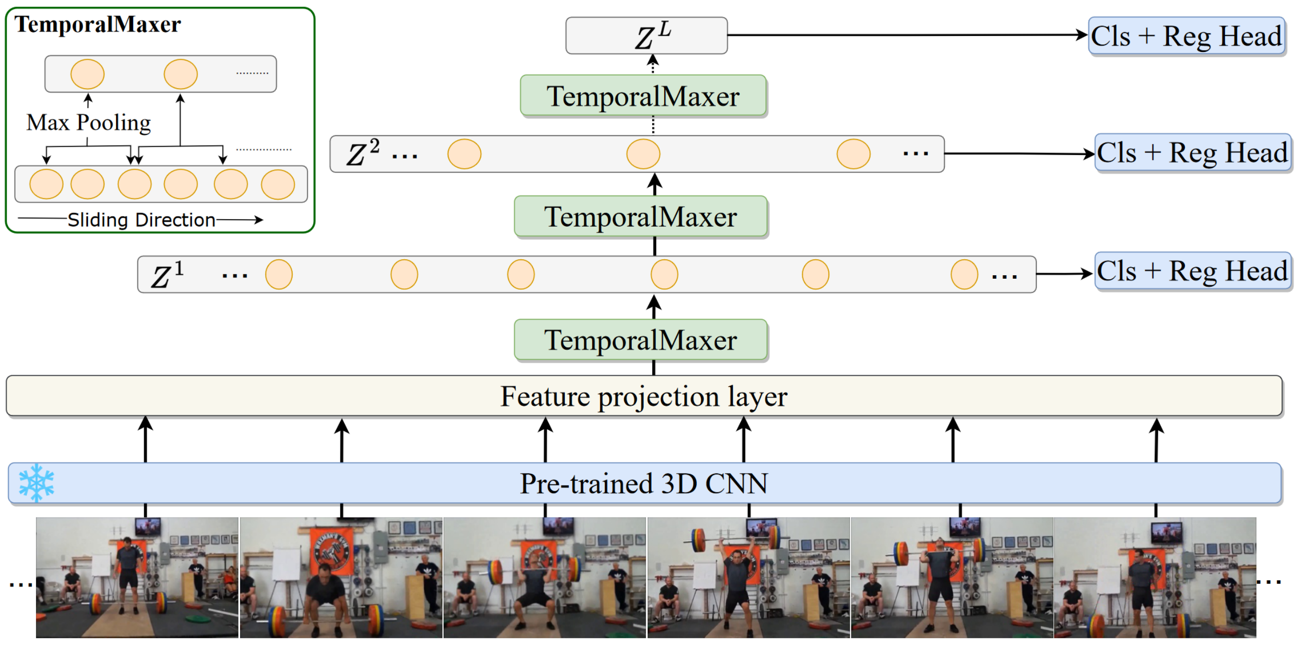

Figure 1. Overview of **TemporalMaxer** taken from [12]. In short, this method applies Max Pooling as a Temporal Context Modeling (TCM) block between layers of a temporal feature pyramid. It aims to enhance significant features from similar video clip embeddings. Initially, it employs a pre-trained 3D CNN to extract features from each clip. Then, it uses a backbone composed of 1D convolutional layers and TemporalMaxer layers to encode these features into a multi-scale feature pyramid. Finally, a simple classification and regression head decodes the feature pyramid, proposing action candidates for each input moment.

mAP scored on the test set. The training duration here refers to the time spent within the training loop, excluding the time consumed during the initialization and other unrelated operations.

To capture the variability in training time and average mAP, the experiment will be repeated five times, and in each iteration of the experiment, five models will be trained, resulting in 25 trained models overall. For each of the five iterations of the experiment, a random seed will be chosen (with a value ranging from 10^9 to $2^{32} - 1$), and the exact values will be found in the public repository of this paper.

3.3.2 Inference performance

The computational efficiency during inference was assessed using videos of varying lengths. Thus, following the practices stated in TemporalMaxer paper [12], the model's performance would be evaluated by calculating the total number of multiply-accumulate operations and the time taken for inference per video. Moreover, analyzing the number of MAC operations can provide further insight into how effectively the hardware is used. Following the same reasoning, memory usage is also tracked. For our calculations, *fvcore* library [38] is used to count the MAC operations, Python's *time.time()* method is used for timing, and PyTorch's *max_memory_allocated* method is used for measuring memory use. The hardware use rate was determined by comparing the true number of MAC operations to the maximum theoretical operations from hardware documentation (one MAC operation equals two floating point operations: add and multiply).

The TemporalMaxer model was evaluated using videos of varying lengths. However, during inference, all the videos are padded to the length indicated by the max_seq_len parameter of the model [15], which has a default value of 2304 for the THUMOS'14 dataset [8]. This indicates that the same computation is done regardless of the videos' original lengths. The model's performance was assessed using random features with shapes corresponding to videos of different increasing lengths (by modifying the parameter to take values from 200 to 3000, with an increment of 200. All inference experiments were conducted independently to avoid unintended correlations in the resulting data. Each experiment was repeated five times. Each repetition used a different random seed, with the exact values available in the project's source code.

4. Experiments

4.1. Setup

The evaluation was performed on the widely-employed THUMOS'14 dataset [8] for TAL-related tasks. This dataset contains 413 unedited and diverse videos spanning 20 action categories of human activities such as sports or musical performances. The dataset is split into a validation set of 213 videos and a test set of 200 videos. Following the methodology presented in the TemporalMaxer paper [12], the model is trained on the validation set and tested on the test set. In this study, the TemporalMaxer model analyzed for data and compute efficiency uses Inflated 3D (I3D) features [25] that were pre-trained on the Kinetics dataset [39]. Training of the model was carried out exclusively using a single GPU Nvidia Tesla V100S 32GB [40] from the DelftBlue supercomputer [14], which was chosen due to the free access provided to Delft University of Technology students and employees. The architecture, code, and hyperparameters of the model remained mostly unaltered from the original paper [12] (the exceptions are noted down below in section 4.3).

Furthermore, the TemporalMaxer model is compared against other SOTA TAL models [37], which include TriDet [16,17], ActionFormer [10,15], and older TAL models such as TadTR [18, 19] (the evaluation for this was effectuated on a different environment than DelftBlue supercomputer). Similar procedures to the ones described in this paper and the same analyses have been conducted on these models with minimal differences.

4.2. Data efficiency

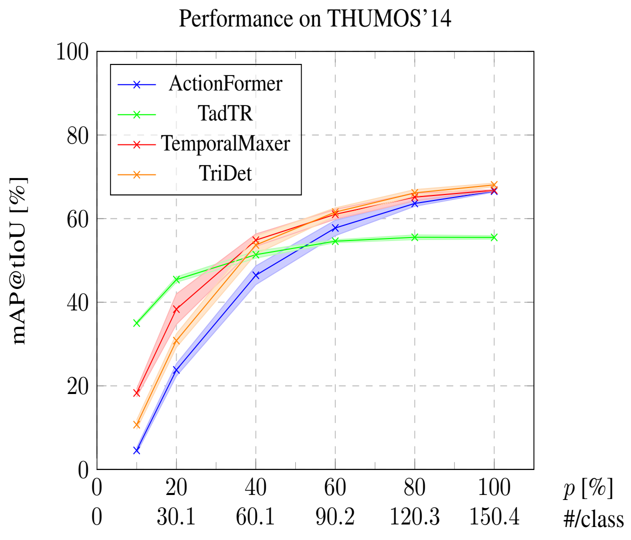

Figure 2. Reported average mAP@tIoU for the tested models on the THUMOS'14 dataset [8].

According to the methodology presented in Section 2.2, algorithm 1 has been applied to assess the efficiency of the TemporalMaxer model in a data-restricted setting. The results are reported on the THUMOS'14 dataset [8] and can be visually observed in Figures 2 and 3.

The training of the models was executed on six different percentages of the training set (the full training set consisted of the validation subset of THUMOS'14 [8]), and it was evaluated on the test set (which coincides with the test subset of THUMOS'14 [8]). Figure 2 shows the results of the TemporalMaxer model compared against the other previously mentioned TAL models (Section 4.1). Several observations can be made from these results. Firstly, the model's accuracy increases with the training dataset percentage, and when reaching 100% it closely replicates the original paper's results [12] with an error of roughly $\pm 0.5\%$. Additionally, it validates the claims of small performance improvements over the ActionFormer [10] model that represents the baseline of TemporalMaxer (for most of the percentages p) [12]. Another interesting aspect is that at the lower percentages p, TemporalMaxer performs significantly better than both ActionFormer [10] and TriDet [16]. However, its performance is significantly lower than that of the older TadTR model [18], which has a completely different architecture and thus a different learning curve (thus, at $p = 100\%$, TemporalMaxer outperforms it by almost 10%). An interesting insight is that with only 40 to 50% of the data, the TemporalMaxer manages to surpass the 50% threshold of performance, which can still be considered "top-tier performance". An increase in the percentage of the data leads to a plateau of around 67.5%, which resembles the claimed performance from the original paper [12].

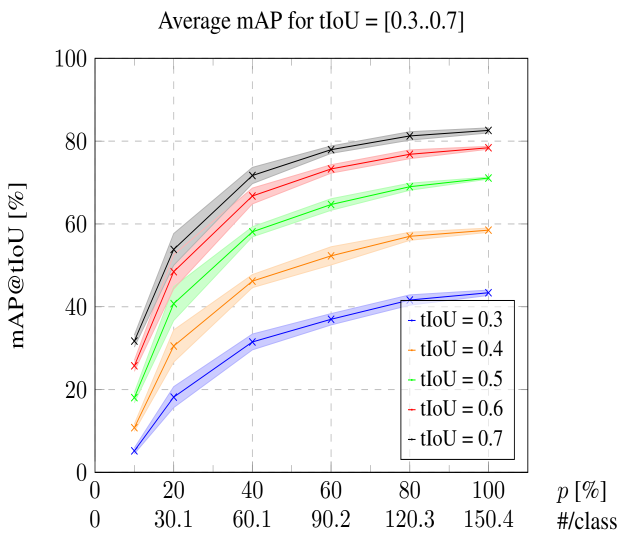

Figure 3. Average mAP for tIoU = [0.3..0.7] on THUMOS'14 [8].

Figure 3 represents the different learning curves that arise from changing the tIoU parameter. At $p = 100\%$, the strictest value for tIoU (0.7) results in performance slightly over 40%, while at the most permissive value of tIoU (0.3), the performance crosses the 80% threshold. Most notably, the results indicate a significant slowdown in the learning rate once the model is trained with over 60% of the THUMOS'14 dataset [8].

4.3. Compute efficiency

4.3.1 Training performance

The outcome of the training experiment (methodology detailed in Subsection 3.3.1) is presented in Tables 1, 2, 3.

Model	Time [s]
TriDet [16]	646.17 ± 26.12
TemporalMaxer [12]	2955.64 ± 1659.98
ActionFormer [10]	866.22 ± 26.97
TadTR [18]	425.72 ± 3.469

Table 1. Training time of the tested models.

Model	Avg. mAP [%]	O. mAP [%]
TriDet [16]	68.07 ± 0.42	69.3
TemporalMaxer [12]	66.96 ± 0.37	67.7
ActionFormer [10]	66.5 ± 0.31	66.8
TadTR [18]	55.3 ± 0.63	56.7

Table 2. Training performance of the tested models. The second column shows the average mAP obtained in the experiments while the third column shows the mAP claimed in the original paper of each model.

Seed	Shortest time	Longest time
4241261284	1676.51	6829.95
2196100728	1403.30	3391.90
1890014188	1275.81	3720.92
3718461004	1216.56	5310.94
3714286883	3979.41	6113.06

Table 3. Ranges of TemporalMaxer [12] training time obtained for each seed.

As anticipated (from the stated performances of the four analyzed models in their original papers, which can be seen in Table 2), the TriDet model [16] achieved the highest mAP at 68.07%, roughly a percent lower than the results documented in the original paper. This minor discrepancy is likely attributable to hardware variations between the initial experiment and our study, which employed resources from the DelftBlue cluster [14]. In terms of performance ranking, TriDet [16] is followed by TemporalMaxer [12], ActionFormer [10], and finally TadTR [18].

We implemented steps to diminish external disruptions in model evaluations. To this end, each model was trained and evaluated 25 times in five groups of five. Each group represents a job submitted to the TU Delft's HPC cluster, DelftBlue [14]. These experiments were carried out at varying times of the day to adjust for fluctuations in cluster load percentages. While these measures were successful for most models, the TemporalMaxer's [12] training duration significantly exceeded our initial estimates, considering the model's similarities with the ActionFormer [10] model. The average training time was approximately 50 minutes, ranging from 1216.56 to 6829.95 seconds. Consequently, the standard deviation was notably high at 1659 seconds compared to the 3 or 26 seconds observed with other models. A complete analysis of these results can be found in Tables 1 and 3.

The key difference in TemporalMaxer is the substitution of the Transformer block in ActionFormer [10] with a MaxPooling block [12]. As mentioned in the original paper and summarized previously, the MaxPooling operation, which the TemporalMaxer uses in its Temporal Context Modeling (TCM) block [12], is conceptually simpler in complexity than ActionFormer's Transformer architecture [10]. While this may result in longer training times, it improves performance, and our experiments prove this fact.

Further research is necessary to definitively ascertain whether these results genuinely reflect the model's inherent properties or if potential external factors linked to the testing environment (DelftBlue [14]) were not entirely isolated. Nonetheless, the close replication of the mAP values may suggest that the findings from this experiment are indeed accurate.

Based on the gathered results, it is clear that while TriDet [16], ActionFormer [10], and TemporalMaxer [12] show comparable accuracy results, the TemporalMaxer model [12] is the least-suited model for an environment that restricts the training time. It yields an accuracy approximately 1% lower for five times the training time. TadTR [18] performs the best in this regard, being 6.5x faster than TemporalMaxer [12], but it comes with a significant drop in performance. A balanced solution is provided by TriDet [16], which is 1.5x slower than TadTR [18] but yields the best performance out of all the models analyzed. Thus, TemporalMaxer [12] is not an optimal solution for training time constraints since it doesn't provide superior performance or rapid training time.

Furthermore, the experiment was conducted exclusively on the THUMOS'14 dataset [8]. As a result, all conclusions mentioned above have limited generalizability but can assist in forming assumptions and hypotheses on the models' efficiency when tested on different datasets.

4.3.2 Inference performance

Figures 5 and 6 showcase the results of the GMACs and inference time & memory experiments. Additionally, figures 4 and 7 contextualize the TemporalMaxer's computational performance with two other models that have similar accuracy-performance on THUMOS'14 [8]: TriDet [16, 17] and ActionFormer [10, 15].

As mentioned in Subsection 3.3.2, these experiments employed random features from the original I3D features [25] extracted from THUMOS'14 [8], with sizes ranging from 200 to 3000 in 200 increments ([15] provides more detailed reasoning for this particular range). These features are benchmarked against the baseline features with a default size of 2304, corresponding to roughly 5 minutes of a THUMOS'14 video [8, 12].

In the original TemporalMaxer paper, it is claimed that the model surpasses the robust baseline, ActionFormer, in performance with 2.8x fewer GMACs and a 3x faster inference speed [12], given a default feature size of 2304.

The first claim holds, as evidenced by Figure 7, where the 2.8x difference is noticeable at size 2304. Unlike TemporalMaxer and TriDet, ActionFormer exhibits a stair-like increase pattern since ActionFormer pads input videos to

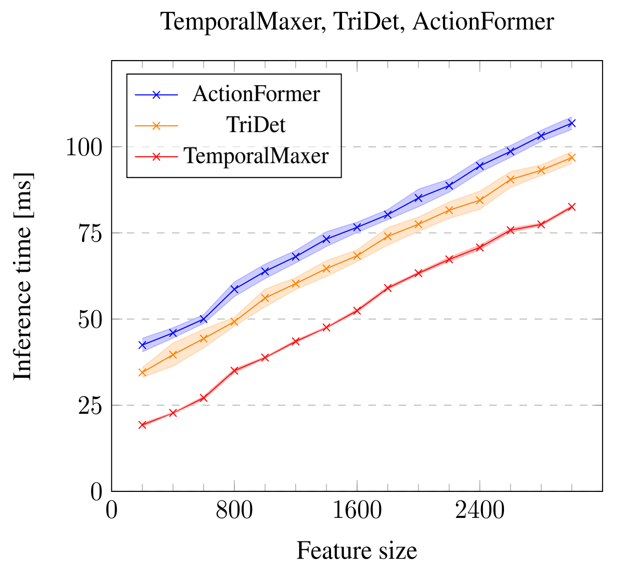

Figure 4. Comparative analysis of average inference times for the ActionFormer [10], TriDet [16], and TemporalMaxer [12] models across varying input feature sizes.

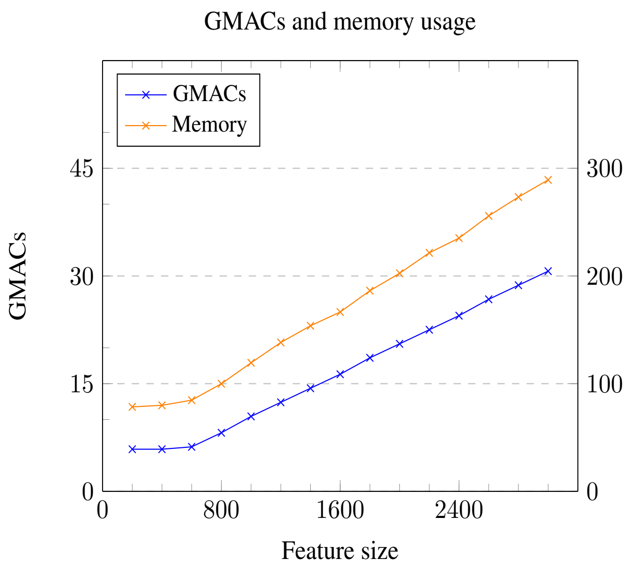

Figure 6. GMACs and memory usage of TemporalMaxer [12]. The right y-axis corresponds to the memory usage in MBs.

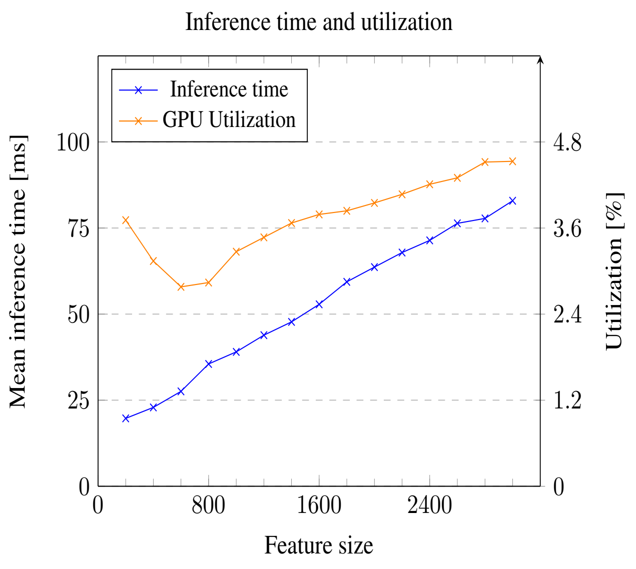

Figure 5. Inference time and utilization of TemporalMaxer [12].

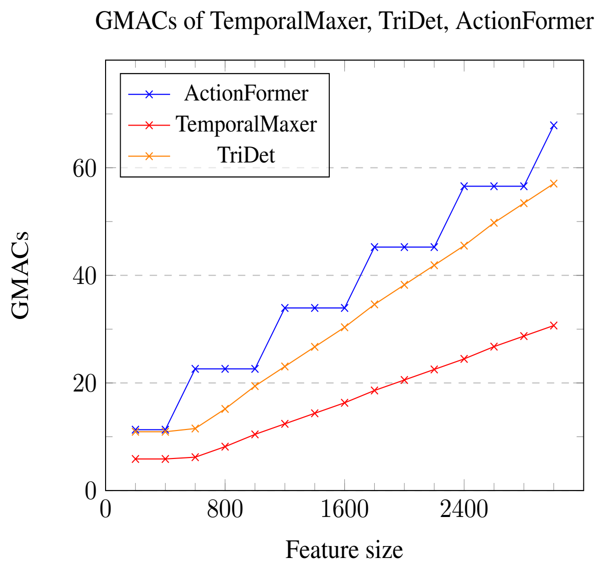

Figure 7. Comparative analysis of GMACs across the TemporalMaxer [12], TriDet [16], and ActionFormer [10] models.

sizes that are multiple of 576, as required by its architecture [10, 15]. In contrast, due to the different TCM employed (TemporalMaxer with its namesake backbone [12], compared to the ActionFormer's transformer backbone [10]), TemporalMaxer exhibits an almost linear increase of GMACs with the lowest slope out of all the three analyzed models (TriDet exhibits a similar increase in shape in GMACs, but with a steeper slope). Moreover, TemporalMaxer and TriDet present constant amounts of GMACs for the smaller feature sizes. This happens likely due to the configurations inherited from the ActionFormer [10] architecture, particularly the *max_seq_len* parameter, which has a lower-bound value of 576, representing the lower boundary for the feature size.

Results to support the second claim (TemporalMaxer having 3x faster inference speed than ActionFormer) are inconclusive. Figure 4 indicates a linear increase in the average inference time for all three models with the rise in feature size, with the difference between the lines remaining relatively constant. Thus, even though TemporalMaxer [12] is approximately 20 seconds faster than the slowest model, ActionFormer [10], it does not demonstrate a 3x difference. The main reason for this discrepancy is a difference in evaluation methodology. The original TemporalMaxer paper computed the average inference time without considering I3D feature extraction and post-processing steps, such as non-maximum suppression (NMS) [12], which depend on the feature size. By strictly analyzing the backbone time (which represents the sole difference between the ActionFormer and TemporalMaxer models), we see that TemporalMaxer backbone time is 2.5 seconds, which is 8x faster than ActionFormer's Transformer TCM that has a backbone time of 20.1 seconds [12]. This explains the constant difference of roughly 20 seconds in Figure 4.

The inference and MACs experiments were conducted on the GPU available on the DelftBlue cluster [14], namely the NVIDIA V100S 32GB GPU, which has a theoretical performance of 8200 GMACs/s [40] used to compute the utilization rate of the model. Figure 5 depicts the GPU's utilization rate and the time taken for inference relative to increasing feature sizes. Both metrics exhibit a linear increase, consistent with other evaluated metrics. The sharp initial drop in utilization rate can be attributed to TemporalMaxer inheriting the lower bound for the feature size of 576 from ActionFormer. As a result, the GMACs value remains constant for sizes below 576, leading to a steep drop in the utilization rate. The utilization rate is calculated as the ratio of GMACs to inference time divided by the maximum theoretical GMACs per second the GPU offers.

The results obtained for the compute efficiency experiments provide insights into finding the optimal hardware specifications needed to run the TemporalMaxer model. TemporalMaxer outperforms the other models significantly in inference speed and GMACs, making it an ideal candidate for computationally constrained environments. To substantiate this, TemporalMaxer requires about 230 MBs of VRAM for features of size 2304 (approximately equivalent to 5 minutes of a THUMOS'14 [8] video), representing a memory usage roughly 4.8x lower than the baseline model, ActionFormer [10, 15].

5. Responsible research

5.1. Reproducibility

Ensuring reproducibility, transparency, and integrity is essential in any type of research, including the Temporal Action Localization (TAL) field. This study upholds these principles by providing a detailed description of the TemporalMaxer model, its architecture, and the methodology employed in the previously-presented experiments. By sharing this information, we enable other researchers to replicate, validate and build upon our work and findings. To assure the reproducibility and transparency of our research, all of our code used to run the experiments and the subsequent results can be found in a publicly available online repository at https://github.com/LeTeutz/data-compute-temporalmaxer, and are licensed with an MIT license. Additionally, documentation is provided for all the steps taken in this research.

Complete reproduction of the results obtained for the data and compute efficiency experiments depends on the type of seed used in each experiment. For fixed-seed experiments, the exact values of the seed can be found in the repository, while for the random-seed experiments, the results we present consist of both a mean and standard deviation value. Reproducibility also depends on the hardware used since the training and inference times may vary significantly based on the available computational resources.

5.2. Ethical considerations

Regarding ethical implications, TAL technology has broad implications in various domains, including video surveillance and commercial applications. TAL's applicability in surveillance scenarios introduces significant ethical concerns, such as balancing public safety and individual privacy. Thus, if deployed without safeguards and regulations, TAL models can potentially infringe upon individual privacy. Therefore, it is crucial to develop responsible practices and guidelines that ensure TAL systems' respectful and ethical use [41].

Our research does not directly improve the current SOTA TAL models or develop a new, improved model; it instead analyses the data and computation efficiency of the TemporalMaxer models and shows that it is possible to achieve decent results even with reduced storage and computational capacity. This, in turn, might spark more research on the feasibility and efficiency of complex TAL models running on Raspberry PIs and other small-sized embedded devices, subsequently improving the current video surveillance capabilities even more. As such, while these advances are promising, it is critical to acknowledge and mitigate the potential privacy and ethical risks associated with increased video surveillance capabilities, ensuring that the progression of technology aligns with the protection of individual rights and societal values.

Furthermore, we underscore the importance of data privacy, favouring publicly accessible or anonymized data sources to ensure individual privacy is upheld. The potential for bias in machine learning models is another paramount ethical consideration. We recommend using diverse, representative training datasets and adversarial training methods to minimize such biases. Furthermore, we acknowledge the importance of ethical considerations and emphasize the use of benchmark datasets, such as THUMOS14 [8], to evaluate the performance of the TemporalMaxer model. By using established datasets, we minimize the need for additional data collection, which could potentially invade privacy or violate ethical guidelines.

This research upholds the principles of reproducibility, transparency, and integrity by sharing detailed information about the model and experimental setup. We also recognize the ethical implications of TAL and emphasize the importance of responsible research practices to mitigate potential privacy concerns and promote the ethical deployment of TAL systems.

6. Conclusion

Temporal action localization (TAL) plays a crucial role in video analysis, aiming to detect and determine the temporal boundaries of specific actions within a video sequence. Despite the great strides that TAL models have made with the adoption of deep learning techniques, these models typically depend on substantial datasets and computationally intensive backbones, like the 1D Convolutional layer in AFSD [42], Graph in G-TAD [26], and Transformers in ActionFormer [10] to encapsulate local and long-term temporal context.

This study analyses the TemporalMaxer model, which introduces a novel backbone by combining a 1D Convolutional layer with a MaxPooling block [12]. We reproduce this state-of-the-art model and evaluate its data and compute efficiency using a well-defined methodology. We compare it to other well-performing TAL models: ActionFormer [10, 15], TriDet [16, 17] and TadTR [18, 19].

To assess the data efficiency of TemporalMaxer, we train the model on subsets of the available training data and measure its performance on the test set. Our results demonstrate that TemporalMaxer achieves satisfactory performance with only 40%-60% of the training data from the THUMOS'14 dataset [8], exhibiting a performance drop of just roughly 10% compared to the original performance. This indicates TemporalMaxer's high degree of generalization in data-limited environments.

Regarding the compute efficiency, we conduct experiments to measure the training time of TemporalMaxer on the THUMOS'14 dataset [8]. The results indicate that TemporalMaxer may not be suitable for situations in which models have to be trained within strict time constraints, as models like TriDet and TadTR outperformed it in such circumstances in terms of both speed and a smaller standard deviation. For additional compute efficiency experiments focused on inference, we examined the number of MACs, video memory usage, and inference time of TemporalMaxer as the input feature sizes grew. Our findings indicate that the model scales linearly with video size, enabling the prediction of inference times on different hardware configurations. Furthermore, TemporalMaxer outperforms other state-of-the-art (SOTA) models, such as TriDet and ActionFormer, in terms of MACs, exhibiting the lowest number of MACs.

While our research provides valuable insights into TemporalMaxer's efficiency, further research is warranted. Future studies could consider strategies like hyperparameter tuning, transfer learning, or semi-supervised learning to optimize performance. Techniques such as temporal downsampling, pruning and quantization, or knowledge distillation may also enhance computational efficiency. These steps would offer deeper insights into TemporalMaxer's performance in real-world, resource-constrained environments.

In conclusion, our study sheds light on the performance of TemporalMaxer in terms of data and compute efficiency. The model exhibits promising data efficiency, performs well with limited training data, and demonstrates linear scalability with video size during inference. These findings offer insights into the practical applicability of TemporalMaxer in resource-constrained scenarios. However, additional research is required to fine-tune its performance and explore other efficiency metrics.

Acknowledgments

I want to express my gratitude to the responsible professor, Dr. Jan van Gemert, and to my supervisors, Robert-Jan Bruintjes, Attila Lengyel, and Ombretta Strafforello, for their valuable insights, constructive feedback, and continuous support. Furthermore, I wish to acknowledge the immense contribution of my teammates. I am grateful for their continuous feedback and encouragement, inspiring and motivating me immensely.

References

[1] Sarvesh Vishwakarma and Anupam Agrawal. A survey on activity recognition and behavior understanding in video surveillance. *The Visual Computer*, 29(10):983–1009, October 2013. 1

[2] Yong Jae Lee, Joydeep Ghosh, and Kristen Grauman. Discovering important people and objects for egocentric video summarization. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1346–1353, June 2012. 1

[3] Yixin Gao, S Swaroop Vedula, Carol E Reiley, Narges Ahmidi, Balakrishnan Varadarajan, Henry C Lin, Lingling Tao, Luca Zappella, Benjamın Béjar, David D Yuh, et al. Jhu-isi gesture and skill assessment working set (jigsaws): A surgical activity dataset for human motion modeling. In *MICCAI workshop: M2cai*, volume 3, 2014. 1

[4] Xiao-Yu Zhang, Haichao Shi, Changsheng Li, Peng Li, Zekun Li, and Peng Ren. Weakly-supervised action localization via embedding-modeling iterative optimization. *Pattern Recognition*, 113:107831, 2021. 1

[5] Hyunjun Eun, Jinyoung Moon, Jongyoul Park, Chanho Jung, and Changick Kim. Temporal filtering networks for online action detection. *Pattern Recognition*, 111:107695, 2021. 1

[6] Kun Xia, Le Wang, Sanping Zhou, Gang Hua, and Wei Tang. Dual relation network for temporal action localization. *Pattern Recognition*, 129:108725, 2022. 1

[7] Huifen Xia and Yongzhao Zhan. A survey on temporal action localization. *IEEE Access*, 8:70477–70487, 2020. 1, 3

[8] Y.-G. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes. http://crcv.ucf.edu/THUMOS14/, 2014. 1, 3, 4, 5, 6, 8, 9

[9] Rizard Renanda Adhi Pramono, Yie-Tarng Chen, and Wen-Hsien Fang. Hierarchical self-attention network for action localization in videos. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 61–70, 2019. 1

[10] Chenlin Zhang, Jianxin Wu, and Yin Li. Actionformer: Localizing moments of actions with transformers, 2022. 1, 2, 5, 6, 7, 8, 9

[11] Yunfeng Yuan, Wenzhu Yang, Zifei Luo, and Ruru Gou. Temporal context modeling network with local-global complementary architecture for temporal proposal generation. *Electronics*, 11(17), 2022. 1

[12] Tuan N Tang, Kwonyoung Kim, and Kwanghoon Sohn. Temporalmaxer: Maximize temporal context with only max pooling for temporal action localization. *arXiv preprint arXiv:2303.09055*, 2023. 1, 2, 3, 4, 5, 6, 7, 8, 9

[13] Tuan N. Tang. Temporalmaxer. https://github.com/TuanTNG/TemporalMaxer, 2023. 1

[14] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022. 1, 5, 6, 8

[15] Jan Warchocki. *Benchmarking Data and Computational Efficiency of ActionFormer on Temporal Action Localization Tasks*. Bachelor's thesis, Delft University of Technology, 2023. 1, 3, 4, 5, 6, 7, 8, 9

[16] Dingfeng Shi, Yujie Zhong, Qiong Cao, Lin Ma, Jia Li, and Dacheng Tao. Tridet: Temporal action detection with relative boundary modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18857–18866, 2023. 1, 2, 5, 6, 7, 9

[17] Alexandru Dămăcuș. *Efficient Video Action Recognition*. Bachelor's thesis, Delft University of Technology, 2023. 1, 5, 6, 9

[18] Xiaolong Liu, Qimeng Wang, Yao Hu, Xu Tang, Shiwei Zhang, Song Bai, and Xiang Bai. End-to-end temporal action detection with transformer. *IEEE Transactions on Image Processing (TIP)*, 2022. 1, 5, 6, 9

[19] Paul Misterka. *Efficient Temporal Action Localization model development practices*. Bachelor's thesis, Delft University of Technology, 2023. 1, 5, 9

[20] Huifen Xia and Yongzhao Zhan. A survey on temporal action localization. *IEEE Access*, 8:70477–70487, 2020. 2

[21] Tianwei Lin, Xiao Liu, Xin Li, Errui Ding, and Shilei Wen. Bmn: Boundary-matching network for temporal action proposal generation, 2019. 2

[22] Shangliang Xu, Xinxin Wang, Wenyu Lv, Qinyao Chang, Cheng Cui, Kaipeng Deng, Guanzhong Wang, Qingqing Dang, Shengyu Wei, Yuning Du, and Baohua Lai. Pp-yoloe: An evolved version of yolo, 2022. 2

[23] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. pages 21–37. Springer International Publishing, 2016. 2

[24] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer, 2016. 2

[25] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017. 2, 5, 6

[26] Mengmeng Xu, Chen Zhao, David S. Rojas, Ali Thabet, and Bernard Ghanem. G-tad: Sub-graph localization for temporal action detection, 2020. 2, 9

[27] Jialin Gao, Zhixiang Shi, Guanshuo Wang, Jiani Li, Yufeng Yuan, Shiming Ge, and Xi Zhou. Accurate temporal action proposal generation with relation-aware pyramid network. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):10810–10817, Apr. 2020. 2

[28] Ahsan Iqbal, Alexander Richard, and Juergen Gall. Enhancing temporal action localization with transfer learning from action recognition. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 1533–1540, 2019. 2

[29] Sauradip Nag, Xiatian Zhu, Yi-Zhe Song, and Tao Xiang. Semi-supervised temporal action detection with proposal-free masking, 2022. 2

[30] Yue Tang, Yawen Wu, Peipei Zhou, and Jingtong Hu. Enabling weakly-supervised temporal action localization from on-device learning of the video stream, 2022. 2

[31] Sauradip Nag, Xiatian Zhu, and Tao Xiang. Few-shot temporal action localization with query adaptive transformer, 2021. 2

[32] Sauradip Nag, Xiatian Zhu, Yi-Zhe Song, and Tao Xiang. Zero-shot temporal action detection via vision-language prompting, 2022. 2

[33] Yizheng Ouyang, Tianjin Zhang, Weibo Gu, and Hongfa Wang. Adaptive perception transformer for temporal action localization, 2022. 2

[34] Phuc Nguyen, Ting Liu, Gautam Prasad, and Bohyung Han. Weakly supervised action localization by sparse temporal pooling network, 2018. 2

[35] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021. 2

[36] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, mar 2021. 2

[37] Papers with Code. Temporal action localization on THUMOS14. https://paperswithcode.com/sota/temporal-action-localization-on-thumos14, n.d. 3, 5

[38] Facebook Research. Fvcore. Available at: https://github.com/facebookresearch/fvcore, 2023. Accessed: 2023-06-25. 4

[39] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017. 5

[40] NVIDIA Corporation. Nvidia v100s tensor core gpu. Available at: https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf, 2020. Accessed on 25/06/2023. 5, 8

[41] Shoshana Zuboff. *The age of surveillance capitalism: The fight for a human future at the new frontier of power: Barack Obama's books of 2019*. Profile books, 2019. 8

[42] Chuming Lin, Chengming Xu, Donghao Luo, Yabiao Wang, Ying Tai, Chengjie Wang, Jilin Li, Feiyue Huang, and Yanwei Fu. Learning salient boundary feature for anchor-free temporal action localization, 2021. 9